

# Auto-encoders: reconstruction versus compression

Yann Ollivier

April 1, 2014

## Abstract

We discuss the similarities and differences between training an auto-encoder to minimize the reconstruction error, and training the same auto-encoder to compress the data via a generative model.

Given a dataset, auto-encoders (for instance, [PH87, Section 8.1] or [HS06]) aim at building a hopefully simpler representation of the data via a hidden, usually lower-dimensional *feature space*. This is done by looking for a pair of maps  $X \xrightarrow{f} Y \xrightarrow{g} X$  from data space  $X$  to feature space  $Y$  and back, such that the reconstruction error between  $x$  and  $g(f(x))$  is small. Identifying relevant features hopefully makes the data more understandable, more compact, or simpler to describe.

Here we take this interpretation literally, by considering auto-encoders in the framework of minimum description length, i.e., data compression via a probabilistic generative model, using the general correspondence between compression and “simple” probability distributions on the data (for instance, [Grü07]). The objective is then to minimize the codelength (log-likelihood) of the data using the features found by the auto-encoder<sup>1</sup>.

This note answers two questions. First, do auto-encoders trained to minimize reconstruction error actually minimize the length of a compressed encoding of the data, or an approximation thereof? Second, if the goal is to compress the data, why should the auto-encoder approach help compared to directly looking for a generative model?

We will see that by adding an information-theoretic term to the reconstruction error, auto-encoders can be trained to minimize a tight upper bound on the codelength (compressed size) of the data. In most situations, directly working with the codelength is difficult (Section 2) hence the interest of such bounds.

In Section 3 we introduce a first, simple bound on codelength based on reconstruction error; it is not tight and is valid only for discrete features. Still it already illustrates how minimizing codelength favors using fewer features.

---

<sup>1</sup>The goal is not to explicitly build an encoding of the data [Grü07]; rather, it is to find a good pair of feature and generative functions that *would* yield a short codelength. If the codelength is known as a function of the parameters of the auto-encoder, it can be used as the training criterion.

In Section 4 we refine the bound from Section 3 and make it valid for general feature spaces. This bound is tight in the sense that it gets arbitrarily close to the actual codelength when the feature and generative functions are inverse to each other in a probabilistic sense. The resulting interpretation of auto-encoders as a tool to optimize codelength is as follows: While the codelength  $L_{\text{gen}}$  depends only on the generative function  $g$  and not on a feature function, we build an upper bound on  $L_{\text{gen}}$  depending on both; optimizing over  $g$  aims at lowering  $L_{\text{gen}}$  by lowering this upper bound, while optimizing over  $f$  aims at making the upper bound more precise.

In Section 5 we treat in more detail the case of continuous features with Gaussian distributions. In addition to a small reconstruction error, minimizing codelength happens to involve a term which favors a more robust reconstruction if noise is introduced in feature space.

In Section 6 we show that optimal compression requires including the variance of the data as additional parameters, especially when various data components have different variances or noise levels. Compression focuses on relative rather than absolute error, minimizing the sum of the *logarithms* of the errors.

Thus, there are differences at several levels between minimizing reconstruction error and minimizing codelength. Note that the discussion here is independent from, and does not replace, strategies used to improve the robustness of auto-encoder training and avoid overfitting, such as contractive or denoising auto-encoders [RVM<sup>+</sup>11, VLL<sup>+</sup>10].

The related preprint [KW13] was pointed to the author after a first version of this text was written. Our Proposition 2 is essentially the same as the result in Section 2.2 of [KW13].

**1 Notation: Auto-encoders, reconstruction error.** Let  $X$  be an input space and  $Y$  be a feature space, usually of smaller dimension.  $Y$  may be discrete, such as  $Y = \{0, 1\}^d$  (each feature present/absent) or  $Y = \{1, \dots, d\}$  (classification), or continuous.

An auto-encoder can be seen as a pair of functions  $f$  and  $g$ , the *feature* function and the *generative* function. The feature function goes from  $X$  to  $Y$  (deterministic features) or to  $\text{Prob}(Y)$  (probability distribution on features), while the generative function goes from  $Y$  to  $X$  or  $\text{Prob}(X)$ .

The functions  $f$  and  $g$  depend on parameters  $\theta_f$  and  $\theta_g$  respectively. Training the parameters via the reconstruction error criterion focuses on having  $g(f(x))$  close to  $x$ , as follows.

Given a feature function  $f: X \rightarrow Y$  and a generative function  $g: Y \rightarrow \text{Prob}(X)$ , define the *reconstruction error* for a dataset  $\mathcal{D} = \{x_1, \dots, x_n\}$  of points of  $X$  as

$$L_{\text{rec}}(x) := -\log g_{f(x)}(x), \quad L_{\text{rec}}(\mathcal{D}) := \mathbb{E}_{x \in \mathcal{D}} L_{\text{rec}}(x) \quad (1)$$

where  $g_y$  is the probability distribution on  $X$  associated with feature  $y$ , and where  $\mathbb{E}_{x \in \mathcal{D}}[\cdot]$  is shorthand for  $\frac{1}{\#\mathcal{D}} \sum_{x \in \mathcal{D}}[\cdot]$ . The case of a deterministic

$g: Y \rightarrow X$  with square error  $\|g(f(x)) - x\|^2$  is recovered by interpreting  $g$  as a Gaussian distribution<sup>2</sup> centered at  $g(f(x))$ . So we will always consider that  $g$  is a probability distribution on  $X$ .

Discrete-valued features can be difficult to train using gradient-based methods. For this reason, with discrete features it is more natural to define  $f(x)$  as a distribution over the feature space  $Y$  describing the law of inferred features for  $x$ . Thus  $f(x)$  will have continuous parameters. If  $f: X \rightarrow \text{Prob}(Y)$  describes a probability distribution on features for each  $x$ , we define the reconstruction error as the expectation of the above:

$$L_{\text{rec}}(x) := -\mathbb{E}_{y \sim f(x)} \log g_y(x), \quad L_{\text{rec}}(\mathcal{D}) := \mathbb{E}_{x \in \mathcal{D}} L_{\text{rec}}(x) \quad (2)$$

This covers the previous case when  $f(x)$  is a Dirac mass at a single value  $y$ .

In Sections 2–4 the logarithms may be in any base; in Sections 5–6 the logarithms are in base  $e$ .

**2 Auto-encoders as generative models.** Alternatively, auto-encoders can be viewed as generative models for the data. For this we assume that we are given (or learn) an elementary model  $\rho$  on feature space, such as a Gaussian or Bernoulli model, or even a uniform model in which each feature is present or absent with probability  $1/2$ . Then, to generate the data, we draw features at random according to  $\rho$  and apply the generative function  $g$ . The goal is to maximize the probability to generate the actual data. In this viewpoint the feature function  $f$  is used only as a prop to learn a good feature space and a good generative function  $g$ .

Given a probability distribution  $p$  on a set  $X$ , a dataset  $(x_1, \dots, x_n)$  of points on  $X$  can be encoded in  $-\sum_i \log_2 p(x_i)$  bits<sup>3</sup>. Let  $\rho \in \text{Prob}(Y)$  be the elementary model on feature space and let  $g: Y \rightarrow \text{Prob}(X)$  be the generative function. The probability to obtain  $x \in X$  by drawing  $y \sim \rho$  and applying  $g$  is

$$p_g(x) := \int_y \rho(y) g_y(x) \quad (3)$$

(where the integral is a sum if the feature space  $Y$  is discrete). Thus minimizing the codelength of the dataset  $\mathcal{D}$  amounts to minimizing

$$L_{\text{gen}}(\mathcal{D}) := \mathbb{E}_{x \in \mathcal{D}} L_{\text{gen}}(x), \quad (4)$$

$$L_{\text{gen}}(x) := -\log p_g(x) = -\log \int_y \rho(y) g_y(x) \quad (5)$$

over  $g$ .

---

<sup>2</sup>While the choice of variance does not influence minimization of the reconstruction error, when working with codelengths it will change the scaling of the various terms in Propositions 1–3. See Section 6 for the optimal variance

<sup>3</sup>Technically, for continuous-valued data  $x$ , the actual compressed length is rather  $-\log_2 p(x) - \log_2 \varepsilon$  where  $\varepsilon$  is the quantization threshold of the data and  $p$  is the probability density for  $x$ . For the purpose of comparing two different probabilistic models  $p$  on the same data with the same  $\varepsilon$ , the term  $-\log_2 \varepsilon$  can be dropped

(This is the codelength of the data knowing the distribution  $\rho$  and the function  $g$ . We do not consider here the problem of encoding the parameters of  $\rho$  and  $g$ .)

The codelength  $L_{\text{gen}}$  does not depend on any feature function  $f$ . However, it is difficult to optimize  $L_{\text{gen}}$  via a direct approach: this leads to working with all possible values of  $y$  for every sample  $x$ , as  $L_{\text{gen}}(x)$  is an integral over  $y$ . Presumably, for each given  $x$  only a few feature values contribute significantly to  $L_{\text{gen}}(x)$ . Using a feature function is a way to explore fewer possible values of  $y$  for a given  $x$ , hopefully those that contribute most to  $L_{\text{gen}}(x)$ .

For instance, consider the gradient of  $L_{\text{gen}}(x)$  with respect to the parameters  $\theta$ :

$$\frac{\partial L_{\text{gen}}(x)}{\partial \theta} = -\frac{\int_y \rho(y) \partial g_y(x) / \partial \theta}{\int_y \rho(y) g_y(x)} = -\frac{\int_y \rho(y) g_y(x) \partial \ln g_y(x) / \partial \theta}{\int_y \rho(y) g_y(x)} \quad (6)$$

$$= -\mathbb{E}_{y \sim p_g(y|x)} \frac{\partial \ln g_y(x)}{\partial \theta} \quad (7)$$

where  $p_g(y|x) = \rho(y)g_y(x) / \int_{y'} \rho(y')g_{y'}(x)$  is the conditional probability of  $y$  knowing  $x$ , in the generative model given by  $\rho$  and  $g$ . In general we have no easy access to this distribution.

Using a feature function  $f$  and minimizing the reconstruction error  $L_{\text{rec}}(x)$  amounts to replacing the expectation under  $y \sim p_g(y|x)$  with an expectation under  $f(x)$  in the above. This is easier to handle, but gives no guarantees about minimizing  $L_{\text{gen}}$  unless we know that the feature function  $f$  is close to the inverse of the generative function  $g$ , in the sense that  $f(x)(y)$  is close to the conditional distribution  $p_g(y|x)$  of  $y$  knowing  $x$ . It would be nice to obtain a guarantee on the codelength based on the reconstruction error of a feature function  $f$  and generative function  $g$ .

Proposition 2 below shows that, given a feature function  $f$  and a generative function  $g$ , the quantity  $L_{\text{rec}}(x) + \text{KL}(f(x) \parallel \rho)$  is an upper bound on the codelength  $L_{\text{gen}}(x)$ . Training an autoencoder to minimize this criterion will thus minimize an upper bound on  $L_{\text{gen}}$ .

Moreover, Proposition 2 shows that the bound is tight when  $f(x)$  is close to  $p_g(y|x)$ , and that minimizing this bound will indeed bring  $f(x)$  closer to  $p_g(y|x)$ . On the other hand, just minimizing the reconstruction error does not, a priori, guarantee any of this.

**3 Two-part codes: explicitly encoding feature values.** We first discuss a simple, less efficient “two-part” [Grü07] coding method. It always yields a codelength larger than  $L_{\text{gen}}$  but is more obviously related to the auto-encoder reconstruction error.

Given a generative model  $g: Y \rightarrow \text{Prob}(X)$  and a prior distribution  $\rho$  on  $Y$ , one way to encode a data sample  $x \in X$  is to explicitly encode a well-chosen feature value  $y \in Y$  using the prior distribution  $\rho$  on features,

then encode  $x$  using the probability distribution  $g_y(x)$  on  $x$  defined by  $y$ . The associated codelength is thus

$$L_{\text{two-part}}(x) := -\log \rho(y) - \log g_y(x) \quad (8)$$

In this section we assume that  $Y$  is a discrete set. Indeed for continuous features, the above does not make sense as encoding a precise value for  $y$  would require an infinite codelength. Continuous features are dealt with in Sections 4 and 5.

We always have  $L_{\text{two-part}}(x) \geq L_{\text{gen}}(x)$ , as  $L_{\text{two-part}}$  uses a single value of  $y$  while  $L_{\text{gen}}$  uses a sum over  $y$  (for discrete features). The difference can be substantial if, for instance, not all feature components are relevant for all  $x$ : using the two-part code, it is always necessary to fully encode the feature values  $y$ .

From an auto-encoder perspective, the feature function  $f$  is used to choose the feature value  $y$  used to encode  $x$ . So if the feature function is deterministic,  $f: X \rightarrow Y$ , and if we set  $y = f(x)$  in the above, the cost of encoding the dataset is

$$\begin{aligned} L_{\text{two-part}}(\mathcal{D}) &= -\mathbb{E}_{x \in \mathcal{D}} (\log \rho(f(x)) + \log g_{f(x)}(x)) \\ &= L_{\text{rec}}(\mathcal{D}) - \mathbb{E}_{x \in \mathcal{D}} \log \rho(f(x)) \end{aligned}$$

involving the reconstruction error and a cross-entropy term between the empirical distribution of features  $f(x)$  and the prior  $\rho$  on feature space. We can further decompose

$$-\mathbb{E}_{x \in \mathcal{D}} \log \rho(f(x)) = \text{KL}(q_f \parallel \rho) + \text{Ent } q_f \quad (9)$$

where  $q_f$  is the empirical distribution of the feature  $f(x)$  when  $x$  runs over the dataset,

$$q_f(y) := \mathbb{E}_{x \in \mathcal{D}} \mathbb{1}_{f(x)=y} \quad (10)$$

and  $\text{KL}(q_f \parallel \rho) = \mathbb{E}_{y \sim q_f} \log(q_f(y)/\rho(y))$  is the Kullback–Leibler divergence between  $q_f$  and  $\rho$ .

If the feature function  $f$  is probabilistic,  $f: X \rightarrow \text{Prob}(Y)$ , the analysis is identical, with the expected two-part codelength of  $x$  being

$$L_{\text{two-part}}(x) = \mathbb{E}_{y \sim f(x)} (-\log \rho(y) - \log g_y(x)) \quad (11)$$

$$= L_{\text{rec}}(x) - \mathbb{E}_{y \sim f(x)} \log \rho(y) \quad (12)$$

Thus we have proved the following, which covers both the case of probabilistic  $f$  and of deterministic  $f$  (by specializing  $f$  to a Dirac mass) on a discrete feature space.

**PROPOSITION 1.** *The expected two-part codelength of  $x \in \mathcal{D}$  and the reconstruction error are related by*

$$L_{\text{two-part}}(\mathcal{D}) = L_{\text{rec}}(\mathcal{D}) - \mathbb{E}_{x \in \mathcal{D}} \mathbb{E}_{y \sim f(x)} \log \rho(y) \quad (13)$$

$$= L_{\text{rec}}(\mathcal{D}) + \text{KL}(q_f \parallel \rho) + \text{Ent } q_f \quad (14)$$

where

$$q_f(y) := \mathbb{E}_{x \in \mathcal{D}} \Pr(f(x) = y) \quad (15)$$

is the empirical distribution of features.

Here are a few comments on this relation. These comments also apply to the codelength discussed in Section 4.

- The reconstruction error in (13) is the *average* reconstruction error for features  $y$  sampled from  $f(x)$ , in case  $f(x)$  is probabilistic. For instance, applying this to neural networks requires interpreting the activities of the abstract layer as probabilities to sample 0/1-valued features on the abstract layer.
- The new cross-entropy term  $-\mathbb{E}_{y \sim q_f} \log \rho(y) = \text{KL}(q_f \parallel \rho) + \text{Ent } q_f$  is an added term to the optimisation problem. The Kullback–Leibler divergence favors feature functions that do actually match an elementary model on  $Y$ , e.g., feature distributions that are “as Bernoulli-like” as possible. The entropy term  $\text{Ent } q_f$  favors parsimonious feature functions that will use fewer feature components if possible, arguably introducing some regularization or sparsity. (Note the absence of any arbitrary parameter in front of this regularization term: its value is fixed by the MDL interpretation.)
- If the elementary model  $\rho$  has tunable parameters (e.g., a Bernoulli parameter for each feature), these come into the optimization problem as well. If  $\rho$  is elementary it will be fairly easy to tune the parameters to find the elementary model  $\rho^*(f)$  minimizing the Kullback–Leibler divergence to  $q_f$ . Thus in this case the optimization problem over  $f$  and  $g$  involves a term  $\text{KL}(q_f \parallel \rho^*(f))$  between the empirical distribution of features and the closest elementary model.

This two-part code is somewhat naive in case not all feature components are relevant for all samples  $x$ : indeed for every  $x$ , a value of  $y$  has to be fully encoded. For instance, with feature space  $Y = \{0, 1\}^d$ , if two values of  $y$  differ in one place and contribute equally to generating some sample  $x$ , one could expect to save one bit on the codelength, by leaving a blank in the encoding where the two values of  $y$  differ. In general, one could expect to save  $\text{Ent } f(x)$  bits on the encoding of  $y$  if several  $y \sim f(x)$  have a high probability to generate  $x$ . We now show that indeed  $L_{\text{two-part}}(x) - \text{Ent } f(x)$  is still an upper bound on  $L_{\text{gen}}(x)$ .

**4 Comparing  $L_{\text{gen}}$  and  $L_{\text{rec}}$ .** We now turn to the actual codelength  $L_{\text{gen}}(x) = -\log p_g(x)$  associated with the probabilistic model  $p_g(x)$  defined by the generative function  $g$  and the prior  $\rho$  on feature space. As mentioned above, it is always smaller than the two-part codelength.

We recall that this model first picks a feature value  $y$  at random according to the distribution  $\rho$  and then generates an object  $x$  according to the distribution  $g_y(x)$ , so that the associated codelength is  $-\log p_g(x) = -\log \int_y \rho(y)g_y(x)$ .

So far this does not depend on the feature function so it is not clear how  $f$  can help in optimizing this codelength. Each choice of  $f$  actually leads to an upper bound on  $L_{\text{gen}}$ : the two-part codelength  $L_{\text{two-part}}$  above is one such bound in the discrete case, and we now introduce a more precise and more general one,  $L_{f\text{-gen}}$ .

We have argued above (Section 2) that for gradient-based training it would be helpful to be able to sample features from the distribution  $p_g(y|x)$ , and it is natural to expect the feature function  $f(x)$  to approximate  $p_g(y|x)$ , so that  $f$  and  $g$  are inverse to each other in a probabilistic sense. The tightness of the bound  $L_{f\text{-gen}}$  is related to the quality of this approximation. Moreover, while auto-encoder training based on the reconstruction error provides no guarantee that  $f$  will get closer to  $p_g(y|x)$ , minimizing  $L_{f\text{-gen}}$  does.

**PROPOSITION 2.** *The codelength  $L_{\text{gen}}$  and reconstruction error  $L_{\text{rec}}$  for an auto-encoder with feature function  $f: X \rightarrow \text{Prob}(Y)$  and generative function  $g: Y \rightarrow \text{Prob}(X)$  satisfy*

$$L_{\text{gen}}(x) = L_{\text{rec}}(x) + \text{KL}(f(x) \parallel \rho) - \text{KL}(f(x) \parallel p_g(y|x)), \quad (16)$$

$$L_{\text{gen}}(\mathcal{D}) = L_{\text{rec}}(\mathcal{D}) + \mathbb{E}_{x \in \mathcal{D}} \text{KL}(f(x) \parallel \rho) - \mathbb{E}_{x \in \mathcal{D}} \text{KL}(f(x) \parallel p_g(y|x)) \quad (17)$$

where  $\rho$  is the elementary model on features, and  $p_g(y|x) = \frac{\rho(y)g_y(x)}{\int_{y'} \rho(y')g_{y'}(x)}$ .

In particular, for any feature function  $f$ , the quantity

$$L_{f\text{-gen}}(\mathcal{D}) := \mathbb{E}_{x \in \mathcal{D}} L_{f\text{-gen}}(x) \quad (18)$$

where

$$L_{f\text{-gen}}(x) := L_{\text{rec}}(x) + \text{KL}(f(x) \parallel \rho) \quad (19)$$

is an upper bound on the codelength  $L_{\text{gen}}(\mathcal{D})$  of the generative function  $g$ .

The proof is by substitution in the right-hand-side of (16). The result holds whether  $Y$  is discrete or continuous.

On a discrete feature space,  $L_{f\text{-gen}}$  is always smaller than the codelength  $L_{\text{two-part}}$  above; indeed

$$L_{f\text{-gen}}(x) = L_{\text{two-part}}(x) - \text{Ent } f(x) \quad (20)$$

as can be checked directly.

The term  $\text{KL}(f(x) \parallel \rho)$  represents the cost of encoding a feature value  $y$  drawn from  $f(x)$  for each  $x$  (encoded using the distribution  $\rho$ ). The last, negative term in (16)–(17) represents how pessimistic the reconstruction error is w.r.t. the true codelength when  $f(x)$  is far from the feature values that contribute most to  $L_{\text{gen}}(x)$ .

The codelength  $L_{\text{gen}}$  depends only on  $g$  and not on the feature function  $f$ , so that the right-hand-side in (16)–(17) is the same for all  $f$  despite appearances. Ideally, this relation could be used to evaluate  $L_{\text{gen}}(\mathcal{D})$  for a given generative function  $g$ , and then to minimize this quantity over  $g$ . However, as explained above, the conditional probabilities  $p_g(y|x)$  are not easy to work with, hence the introduction of the upper bound  $L_{f\text{-gen}}$ , which does depend on the feature function  $f$ .

Minimizing  $L_{f\text{-gen}}$  over  $f$  will bring  $L_{f\text{-gen}}$  closer to  $L_{\text{gen}}$ . Since  $L_{\text{gen}}(\mathcal{D}) = L_{f\text{-gen}}(\mathcal{D}) - \mathbb{E}_{x \sim \mathcal{D}} \text{KL}(f(x) \| p_g(y|x))$ , and since  $L_{\text{gen}}$  does not depend on  $f$ , minimizing  $L_{f\text{-gen}}$  is the same as bringing  $f(x)$  closer to  $p_g(y|x)$  on average. Thus, in the end, an auto-encoder trained by minimizing  $L_{f\text{-gen}}$  as a function of both  $f$  and  $g$  will both minimize an upper bound on the codelength  $L_{\text{gen}}$  and bring  $f(x)$  close to the “inverse” of  $g$ .

This also clarifies the role of the auto-encoder structure in minimizing the codelength, which does not depend on a feature function: Optimizing over  $g$  aims at actually reducing the codelength by decreasing an upper bound on it, while optimizing over  $f$  will make this upper bound more precise.

One can apply to  $L_{f\text{-gen}}$  the same decomposition as for the two-part codelength, and write

$$L_{f\text{-gen}}(\mathcal{D}) = L_{\text{rec}}(\mathcal{D}) + \text{KL}(q_f \| \rho) + \text{Ent } q_f - \mathbb{E}_{x \sim \mathcal{D}} \text{Ent } f(x) \quad (21)$$

where as above  $q_f = \mathbb{E}_{x \sim \mathcal{D}} f(x)$  is the empirical feature distribution. As above, the term  $\text{KL}(q_f \| \rho)$  favors feature distributions that match a simple model. The terms  $\text{Ent } q_f$  and  $\mathbb{E}_{x \sim \mathcal{D}} \text{Ent } f(x)$  pull in different directions. Minimizing  $\text{Ent } q_f$  favors using fewer features overall (more compact representation). Increasing the entropy of  $f(x)$  for a given  $x$ , if it can be done without impacting the reconstruction error, means that more features are “indifferent” for reconstructing  $x$  and do not have to be encoded, as discussed at the end of Section 3.

The “auto-encoder approximation”  $x' = x$  from [AO12, Section 2.4] can be used to define another bound on  $L_{\text{gen}}$ , but is not tight when  $f(x) \approx p_g(y|x)$ .

## 5 Continuous-valued features with deterministic feature function.

Proposition 2 cannot be directly applied to a deterministic feature function  $f: X \rightarrow Y$  with values in a continuous space  $Y$ . In the continuous case, the reconstruction error based on a single value  $y \in Y$  cannot control the codelength  $L_{\text{gen}}(x)$ , which involves an integral over  $y$ . In the setting of Proposition 2, a deterministic  $f$  seen as a probability distribution is a Dirac mass at a single value, so that the term  $\text{KL}(f(x) \| \rho)$  is infinite: it is infinitely costly to encode the feature value  $f(x)$  exactly.

This can be overcome in at least two ways:

1. In many situations, a continuous feature value  $y$  can be seen as a probability distribution under an underlying space  $Z$ , namely,  $Y =$



$\text{Prob}(Z)$ . This is the case, for instance, for neural networks, for which the activities of the innermost layer lie in  $[0; 1]$  and can be seen as Bernoulli probabilities for discrete-value binary features. In this case, Proposition 2 can be applied to  $f(x)$  seen as a probability distribution over the feature space  $Z$ . Note that the reconstruction error  $L_{\text{rec}}$  in (16) becomes an expectation over samples  $z \sim f(x)$ .

2. One can try to control the reconstruction error  $\log g_y(x)$  not only at  $y = f(x)$ , but in some neighborhood of  $f(x)$ , so that intuitively  $f(x)$  has to be encoded only with smaller accuracy. Thus, having  $g_y(x)$  “as flat as possible” around  $y = f(x)$  will help. A single value  $y \in Y$  can be seen as parametrizing a probability distribution  $\pi_y$  over  $Y$  itself, such as a Gaussian distribution centered at  $y = f(x)$  with a small covariance matrix  $\Sigma$ . Proposition 2 applied to  $\pi_{f(x)}$  instead of  $f(x)$  will then provide an upper bound on  $L_{\text{gen}}$  based, not on the reconstruction error at  $y$ , but on the reconstruction error in a small neighborhood around  $y$ . A good choice of  $\Sigma$  leads to tighter upper bounds; the optimal  $\Sigma$  is the Hessian of  $-\log(\rho(y)g_y(x))$  at  $y$ . This leads to the following.

**PROPOSITION 3.** *Let  $f: X \rightarrow Y$  be a deterministic feature function with values in  $Y = \mathbb{R}^d$ . Let  $L_{\mathcal{N}f\text{-gen}}$  be the upper bound obtained from Proposition 2 by using a normal distribution  $\mathcal{N}(f(x), \Sigma)$  as the feature distribution instead of  $f(x)$ . Then for small covariance matrix  $\Sigma$  we have*

$$L_{\mathcal{N}f\text{-gen}}(x) \approx -\log g_{f(x)}(x) - \log \rho(f(x)) - \frac{1}{2} \log \det \Sigma + \frac{1}{2} \text{Tr}(\Sigma H) - \frac{d}{2} (1 + \log 2\pi) \quad (22)$$

which is thus an approximate upper bound on  $L_{\text{gen}}(x)$ . Here  $H$  is the Hessian of  $-\log(g_y(x)\rho(y))$  at  $y = f(x)$ .

The choice  $\Sigma = H^{-1}$  (provided  $H$  is positive, which is the case close to the minimum) is optimal and yields

$$L_{\mathcal{N}f\text{-gen}}(x) \approx -\log g_{f(x)}(x) - \log \rho(f(x)) + \frac{1}{2} \log \det H - \frac{d}{2} \log 2\pi \quad (23)$$

as an approximate upper bound on  $L_{\text{gen}}(x)$ .

In addition to the reconstruction error  $-\log g_{f(x)}(x)$  at  $f(x)$  and to the encoding cost  $-\log \rho(f(x))$  under the elementary model, this codelength bound involves the reconstruction error at points  $y$  around  $f(x)$  through the Hessian of the reconstruction error. Minimizing this bound will favor points where the error is small in the widest possible region around  $f(x)$ . This presumably leads to more robust reconstruction with noisy features.

### PROOF OF PROPOSITION 3.

Using  $y \sim \mathcal{N}(f(x), \Sigma)$  in Proposition 2, the reconstruction error  $L_{\text{rec}}(x)$  is  $-\mathbb{E}_y \log g_y(x)$ . Using a second-order Taylor expansion of  $\log g_y(x)$  around

$y = f(x)$ , and using that  $\mathbb{E}_{z \sim \mathcal{N}(0, \Sigma)}(z^\top M z) = \text{Tr}(\Sigma M)$  for any matrix  $M$ , we find  $L_{\text{rec}}(x) \approx -\log g_{f(x)}(x) + \frac{1}{2} \text{Tr}(\Sigma H_g)$  where  $H_g$  is the Hessian of  $-\log g_y(x)$  at  $y = f(x)$ . By a similar argument the term  $\text{KL}(\mathcal{N}(f(x), \Sigma) \parallel \rho)$  is approximately  $-\text{Ent} \mathcal{N}(f(x), \Sigma) - \log \rho(f(x)) + \frac{1}{2} \text{Tr}(\Sigma H_\rho)$  with  $H_\rho$  the Hessian of  $-\log \rho(y)$  at  $y = f(x)$ . Thus the bound  $L_{f\text{-gen}}(x)$  is approximately  $-\log g_{f(x)}(x) - \log \rho(f(x)) - \text{Ent} \mathcal{N}(f(x), \Sigma) + \frac{1}{2} \text{Tr}(\Sigma H)$  with  $H = H_g + H_\rho$ . The result follows from  $\text{Ent} \mathcal{N}(f(x), \Sigma) = \frac{1}{2} \log \det \Sigma + \frac{d}{2} (1 + \log 2\pi)$ .

Substituting  $\Sigma = H^{-1}$  yields the second estimate. Let us prove that this choice is optimal. We have to minimize  $-\log \det \Sigma + \text{Tr}(\Sigma H)$  over  $\Sigma$ . We have  $\text{Tr}(\Sigma H) = \text{Tr}(H^{1/2} \Sigma H^{1/2})$ . Since  $H^{1/2} \Sigma H^{1/2}$  is symmetric we can decompose  $H^{1/2} \Sigma H^{1/2} = O^\top D O$  with  $O$  orthogonal and  $D$  diagonal. Then  $\text{Tr}(\Sigma H) = \text{Tr}(O^\top D O) = \text{Tr}(D)$ . Moreover,  $\log \det \Sigma = \log \det(H^{-1/2} O^\top D O H^{-1/2}) = -\log \det H + \log \det D$  so that  $-\log \det \Sigma + \text{Tr}(\Sigma H) = \log \det H - \log \det D + \text{Tr}(D) = \log \det H + \sum_k (d_k - \log d_k)$  with  $d_k$  the entries of  $D$ . The function  $z \mapsto z - \log z$  is convex on  $\mathbb{R}_+$  with a unique minimum at  $z = 1$ , so this is minimal if and only if  $D = \text{Id}$ , i.e.,  $\Sigma = H^{-1}$ .  $\square$

When the reconstruction error  $-\log g(h)(x)$  is of the form  $\ell(\hat{x}, x)$  for some approximation  $\hat{x}$  of  $x$  (such as the square loss), we can understand the effect of the term  $\frac{1}{2} \log \det H$  as follows. Under the standard Gauss–Newton approximation we have  $H \approx \frac{\partial \hat{x}}{\partial y}^\top \frac{\partial^2 \ell(\hat{x}, x)}{\partial \hat{x}^2} \frac{\partial \hat{x}}{\partial y}$  when  $\hat{x}$  is not too far from  $x$  on average. Thus, small derivatives  $\frac{\partial \hat{x}}{\partial y}$  will be favored in the directions where the loss is most sensitive, hence a more robust reconstruction  $\hat{x}$ .

Still, the term  $\frac{1}{2} \log \det H$  is a fitness term, not a regularization term. By itself, it favors fitting more closely to the dataset. For instance, with one-dimensional data samples  $x \in [0; 1]$ , with features  $y \in [0; 1]$  and a uniform prior  $\rho$ , the optimal reconstruction (shortest codelength) is when the map  $y \rightarrow x$  is piecewise constant so that the uniform distribution on  $y$  is exactly mapped to the dataset distribution: indeed the derivatives  $\frac{\partial \hat{x}}{\partial y}$  vanish. Thus, regularization has to come from other mechanisms (dimension of the model, specific regularization terms such as denoising or contractive auto-encoders...). In this example, the piecewise constant generative function is generally not feasible within a reasonable model class, and the term  $\frac{1}{2} \log \det H$  may make the reconstruction function flatter in the regions with a high density of sample points, so that the image of the uniform distribution on  $y$  will be a distribution concentrated around the data points.

**6 Variance of the output, and relative versus absolute error.** A final, important choice when considering auto-encoders from a compression perspective is whether or not to include the variance of the output as a model parameter. While minimizing the reconstruction error usually focuses on absolute error, dividing the error by two will reduce codelength by one bit whether the error is large or small. This works out as follows.

Consider a situation where the outputs are real-valued (e.g., image). The usual loss is the square loss  $L = \sum_n \sum_i (x_n^i - \hat{x}_n^i)^2$  where  $n$  goes through all samples and  $i$  goes through the components of each sample (output dimension), the  $x_n$  are the actual data, and the  $\hat{x}_n$  are the reconstructed data computed from the features  $y$ .

This square loss is recovered as the log-likelihood of the data over a Gaussian model with fixed variance  $\sigma$  and mean  $\hat{x}_n$ :

$$L_{\text{rec}}(\mathcal{D}) = -\mathbb{E}_{x \in \mathcal{D}} \log g_{\hat{x}}(x) = \mathbb{E}_{x \in \mathcal{D}} \sum_i \left( \frac{(x^i - \hat{x}^i)^2}{2\sigma^2} + \log \sigma + \frac{1}{2} \log 2\pi \right) \quad (24)$$

For any fixed  $\sigma$ , the optimum is the same as for the square loss above.

Incorporating a new parameter  $\sigma_i$  for the variance of the  $i$ -th component into the model may make a difference if the various output components have different scales or noise levels. The reconstruction error becomes

$$L_{\text{rec}}(\mathcal{D}) = \sum_i \mathbb{E}_{x \in \mathcal{D}} \left( \frac{(x^i - \hat{x}^i)^2}{2\sigma_i^2} + \log \sigma_i + \frac{1}{2} \log 2\pi \right) \quad (25)$$

which is now to be optimized jointly over the functions  $f$  and  $g$ , and the  $\sigma_i$ 's. The optimal  $\sigma_i$  for a given  $f$  and  $g$  is the mean square error of component  $i$ ,

$$\sigma_i^{*2} = E_i := \mathbb{E}_{x \in \mathcal{D}} \left( (x^i - \hat{x}^i)^2 \right) \quad (26)$$

so with this optimal choice the reconstruction error is

$$L_{\text{rec}}(\mathcal{D}) = \sum_i \left( \frac{1}{2} + \frac{1}{2} \log E_i + \frac{1}{2} \log 2\pi \right) \quad (27)$$

and so we have to optimize

$$L_{\text{rec}}(\mathcal{D}) = \frac{1}{2} \sum_i \log E_i + \text{Cst} \quad (28)$$

that is, the sum of the *logarithms* of the mean square error for each component. (Note that this is not additive over the dataset: each  $E_i$  is an average over the dataset.) Usually the sum of the  $E_i$  themselves is used. Thus, including the  $\sigma_i$  as parameters changes the minimization problem by focusing on *relative* error, both for codelength and reconstruction error.

This is not cancelled out by normalizing the data: indeed the above does not depend on the variance of each component, but on the mean square error, which can vary even if all components have the same variance (if some components are harder to predict).

This must be used with caution, in particular when some errors become close to 0 (the log tends to  $-\infty$ ). Indeed, optimizing this objective function means that one prediction with an accuracy of 100 digits over one single data sample can balance out 100 bad predictions on other samples. This

is only relevant if the data are actually precise up to 100 significant digits. In practice an error of 0 only means that the actual error is below the quantization level  $\varepsilon$ . Thus, numerically, we might want to consider that the smallest possible square error is  $\varepsilon^2$ , and to optimize  $\sum_i \log(E_i + \varepsilon^2)$  for data quantized up to  $\varepsilon$ .

Incidentally, when working with Propositions 2 and 3, changing  $\sigma$  can have a large influence: it changes the relative scaling of the reconstruction error term  $L_{\text{rec}}$  w.r.t. the remaining information-theoretic terms. Choosing the optimal  $\sigma_i$  as described here fixes this problem and makes all terms homogeneous.

Intuitively, from the minimum description length or compression viewpoint, dividing an error by 2 is an equally good move whether the error is small or large (one bit per sample gained on the codelength). Still, in a specific application, the relevant loss function may be the actual sum of square errors as usual, or a user-defined perceptual error. But in order to find a good representation of the data as an intermediate step in a final, user-defined problem, the compression point of view might be preferred.

## References

- [AO12] Ludovic Arnold and Yann Ollivier. Layer-wise training of deep generative models. Preprint, arXiv:1212.1524, 2012.
- [Grü07] Peter D. Grünwald. *The minimum description length principle*. MIT Press, 2007.
- [HS06] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.
- [KW13] Diederik P. Kingma and Max Welling. Stochastic gradient vb and the variational auto-encoder. Preprint, arXiv:1312.6114, 2013.
- [PH87] David C. Plaut and Geoffrey Hinton. Learning sets of filters using back-propagation. *Computer Speech and Language*, 2:35–61, 1987.
- [RVM<sup>+</sup>11] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*, pages 833–840, 2011.
- [VLL<sup>+</sup>10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.